The pImpl Idiom Solutions

Copyright © James Raynard 2021

Interface/Implementation Separation

- Why is separating the interface of a class from its implementation important in object-oriented programming?
  - It prevents users of the class from accessing the implementation of the class
  - This reduces coupling between the users' code and the implementation code
  - e.g. if an implementation detail changes, the users do not need to modify their code

Copyright © James Raynard 2021

C++ class organization

- Why is it difficult to completely separate the interface and implementation of a C++ class?
  - Users of a C++ class need to include a header file which contains the class definition
  - This exposes the names of private and protected data members, the signatures of private and protected member functions, and the definition of inline functions
- What problems can this cause when compiling large applications?
  - If any changes are made to the header, all the users of the class need to recompile their code
  - This applies even if the change does not affect the client

Copyright © James Raynard 2021

The Handle-Body pattern

- What is the the Handle-Body pattern?
  - The class is divided into two parts
  - The Handle contains the interface (public member functions)
  - The Body contains the implementation of the class
  - The member functions of the Handle call the corresponding member functions of the Body

Copyright © James Raynard 2021

The pImpl Idiom

- What is the pImpl idiom?
  - The pImpl idiom implements the Handle-Body pattern by giving the Handle class a private data member which is a pointer to a Body object
  - This is a "pointer to the implementation" (pImpl)
  - A Body object is allocated in the Handle's constructor and released in the Destructor
  - The member functions of the Handle forward their arguments to the corresponding member functions of the Body

Copyright © James Raynard 2021

The pImpl Idiom

- How does the pImpl idiom help with the separation problem?
  - Since Handle has a pointer to a Body, it only needs a declaration of Body, not the entire class definition
  - It therefore does not need to include Body's header
  - Since the clients only interact with a Handle, they do not need to include Body's header either
  - Clients will not be aware of any changes in the Body implementation (provided the Handle interface is unchanged)
  - Clients do not need to recompiled when the Body implementation changes

Copyright © James Raynard 2021

Pros and cons

- What other advantages does the pImpl idiom have when compiling and distributing large applications?
  - Since the Body implementation is separate from the rest of the program, it can be compiled into a DLL/shared object
  - (A DLL/shared object is loaded by the program when it starts executing)
  - This means that changes to the Body implementation only require recompiling the DLL
  - If there is a change to the Body interface which affects the Handle's implementation, but not the Handle's interface, client code does not need to be re-compiled (only re-linked)

Copyright © James Raynard 2021

Pros and cons

- What disadvantages does it have?
  - Requires an extra memory allocation
  - Member functions calls require a pointer dereference
  - Adds complexity

Copyright © James Raynard 2021

pImpl with unique_ptr

- Write a simple program that uses the pImpl idiom with unique_ptr
- Your program should organize the code in the usual way (class definition in header file, member function definitions in source file)

Copyright © James Raynard 2021